intel®

APPLICATION NOTE

AP-312

October 1988

Creating Macros for EPLD Designs

DANIEL E. SMITH
PROGRAMMABLE LOGIC APPLICATIONS
INTEL CORPORATION

Order Number: 292040-002

INTRODUCTION

The iPLS II (Intel Programmable Logic Software II) Logic Optimizing Compiler includes a Macro Expander that supports the use of macros in EPLD designs. These macros can include TTL and EPLD custom macros available from Intel, or proprietary macros developed by a user. This application note shows how to create user-defined macros and how to build macro libraries with Intel's Macro Librarian, an optional software package for use with iPLS II. A design example also shows creation of a user-defined macro and its use in an ADF (Advanced Design File). Detailed information on using the TTL Macros in iPLS II ADFs are described in a companion application note, AP-311 "Using Macros in EPLD Designs", Order Number: 292039. This application note concentrates on creating macros; it assumes that you have read and understood the discussion on using macros in AP-311.

OVERVIEW

iPLS II allows designers to include macro calls in design files to implement common circuit functions. Macros calls are subsequently expanded by the LOC (Logic Optimizing Compiler) into the ADF network and/or equation entries required to perform the desired functions. Macros can be connected together or used in conjunction with standard iPLS II EPLD primitives.

By following the macro file format described in this note, users can also create their own proprietary macros with an ASCII text editor. These macro files can then be stored in user-defined libraries by using Intel's Macro Librarian software. User-defined macros can be called from ADFs created by a text editor or by schematic capture software that supports user-defined symbols and that outputs in ADF format. User-defined macros can optimize development of EPLD designs by modularizing the design process and by allowing the design process to proceed at a higher level than with EPLD primitives alone. iPLS II support for user-defined macros (see in Figure 1) includes the following:

- MLIB, the optional iPLS II Macro Librarian for creating macro libraries from individual user-defined macro files.
- a Macro Expander in the LOC that expands macro calls in ADFs with the contents of the corresponding macros from libraries.

This application note describes how to create macro files, store them in libraries with MLIB, and shows how to call them from ADFs created by a text editor. *For information on creating user-defined macro symbols with schematic capture packages, refer to the appropriate manual for the schematic capture package you are using.* SCHEMA II-PLD available from Intel supports user-defined symbols and outputs in ADF format.

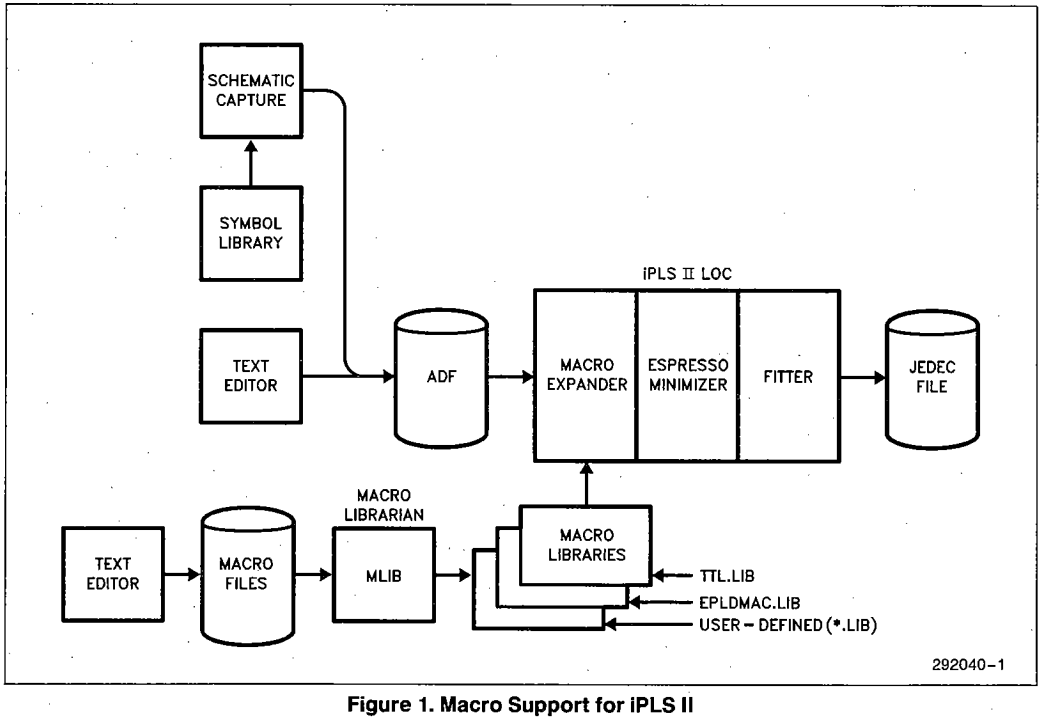

Figure 1. Macro Support for iPLS II

(SCHEMA II-PLD is based on SCHEMA II from Omation, Inc. The Intel EPLD Design Manager, also available from Intel, allows existing SCHEMA II users to design with EPLDs and macros.)

MACRO FILES

This section describes iPLS II macro files. **User-defined macro files must follow the guidelines presented here** to be successfully processed by the Macro Librarian (MLIB) and expanded by the iPLS II LOC Macro Expander.

Macro filenames follow DOS conventions. It is recommended that macro filenames end with the extension .DEV, which is the default for MLIB. Only one macro can be contained in a macro file. Macro files are comprised of three sections:

- Header
- Network Section
- Equation Section

All macro files must end with the literal "ENDEF". Figure 2 shows a sample macro file for a proprietary part (16207), a "black box" containing random logic.

```
16207(A,B,C,D,E,F,U,V,W,X,Y,Z)
DEFAULT:(GND,GND,GND,VCC,VCC,VCC,,,,,,)

EQUATIONS:
        U = /(A * B);
        V = /(/E * A * B);
        W = /(D * C * A * /E);
        X = /(/D * E);
        Y = /(F * D * A);
        Z = F * /E;

ENDEF
                                    292040-2
```

Figure 2. Sample Macro File for "Black Box" (16207.DEV)

Header

Headers for macro files contain two lines. The first line includes the name of the macro function and a list of inputs and outputs for the macro. The second line contains defaults for the device.

The name of the macro can be a device number (16207, 83546, etc.), function name (ADDRCNT, CMDLO, etc.), or any name up to eight characters long. No spaces or comments precede the name.

Inputs and Outputs follow immediately after the macro name and are enclosed in parentheses. I/O signal names may be up to eight characters long, but may not contain pin numbers. For user-defined macros, signals may be listed in any order desired. For example, any of the following entries are legal:

16207 (A,B,C,D,E,F,U,V,W,X,Y,Z)

16207 (B,D,A,R,Z,U,W,C,F,X,E,Y)

16207 (Z,Y,X,W,V,U,F,E,D,C,B,A)

Note that this **first line of the header forms the template used to call the Macro from the ADF.** The Macro Expander connects ADF nodes in the macro call to I/O signals in the macro file on the basis of **position**, not on the basis of node name.

The second line in the header specifies defaults for inputs (VCC or GND) in cases where those signals are left unconnected. The DEFAULT: line must be included in the macro definition file, even when no defaults are used in the ADF. The keyword DEFAULT: is the first entry in this line. The default values for all signals follow immediately and are enclosed in parentheses. Input defaults may be VCC or GND. The position of the default value corresponds to the signal listed in the previous line.

Defaults for outputs are blank, but a comma (,) must be present (place holder) for each output signal except the last. For example, the 16207 black box contains six inputs (A through F) and six outputs (U through Z). The first two lines for this macro might be:

```
16207 (A,B,C,D,E,F,U,V,W,X,Y,Z)
DEFAULT: (GND,GND,GND,VCC,VCC,VCC,,,,,,)
```

Defaults for inputs A through C are GND; defaults for inputs D through F are VCC. Defaults for the outputs are not specified, but the comma denotes the positions for those signals.

Defaults should be chosen with care. Clears, Presets, Loads, etc. should be disabled in most cases. Enables should be enabled. Input defaults can also be left blank as long as those inputs are connected to nodes in the ADF that calls the macro, but it is recommended that they be specified in the macro file.

Network Section

The NETWORK: section lists the EPLD primitives used to implement the desired functions. The Network Section follows ADF syntax rules. As far as possible, the macros should be implemented in equations to eliminate concern about feedbacks and output enables. In the case of a circuit that requires macrocell registers, the feeback-only form of the primitive should be used so that the Macro Expander can make the correct pin connections. The following example shows this:

```
OUT1 = NORF (INd,CLK,GND,GND)
```

During processing, the Macro Expander connects the feedback to an output (if necessary) and supplies the required output enable node name. The Macro Expander also eliminates unneeded Network and Equations entries if they are not used by an ADF.

If no network entries are required (i.e., a macro implemented entirely in equations), the entire Network section may be omitted, including the keyword NETWORK:. In many cases, equations alone can implement the desired functions.

Equations Section

The EQUATIONS: section lists the Boolean equations for the desired functions and follows ADF syntax rules, with one exception; intermediate equations are not permitted in macro files. If no equation entries are required (i.e., a macro implemented entirely in the Network Section), the entire Equation section may be omitted, including the keyword EQUATIONS:.

Comments and White Space

Comments can be placed anywhere in a macro file except before the name and signals on the first line. Comments must be enclosed in percent signs, as follows:

```
% THIS IS A SAMPLE COMMENT %
```

White space can appear on any line except the first two lines.

MACRO LIBRARIAN

The Macro Librarian (MLIB) is an optional software package that combines individual macro files into macro libraries. These libraries are in turn used by the LOC Macro Expander. MLIB can be invoked from the command line, from command files, or from a combination of both. Figure 3 shows a block diagram of the Macro Librarian.

Syntax for MLIB command lines is as follows:

MLIB [-options] [@cmdfile] [file1 file2 . . .]
<Enter>

-d directory. Displays directory information for the library being created.

-v verbose. Print status during processing. When not specified, status messages are suppressed.

-l lib list. Lists the contents of existing macro library to console. This option may not be used while building a library.

-o lib name of the target macro library. MACRO.LIB is the default when no name is specified. TTL.LIB, EPLDMAC.LIB, and INTEL.LIB are reserved for Intel libraries and may not be used.

-s string include version stamp in macro library. The version string can be up to 7 characters long. "V1.00" is the default stamp.

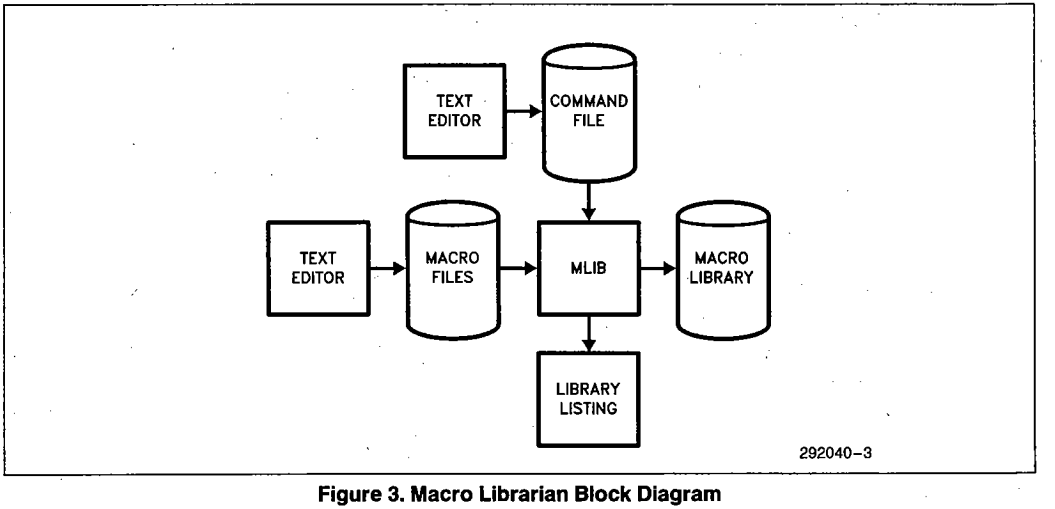

Figure 3. Macro Librarian Block Diagram

-c string include copyright string in macro library. The copyright string can be up to 61 characters long and, if blanks are used, must be contained in quotation marks, for example, "texta textb".

@cmdfile name of command file. The command file can include options and macro filenames. The @ symbol must precede the filename.

file1 . . . name of device files to be included in the macro library. Separate files by spaces.

For example, the following command line:

MLIB -v -s 2.00 -o USER.LIB @USERLIST <Enter>

creates a library called USER.LIB that includes all the individual macro files contained in the command file USERLIST. MLIB displays status messages as it processes the macro files in USERLIST (-v). The library is created as version 2.00 (-s).

Macro library filenames follows DOS conventions and should end with the extension .LIB to be recognized by the Macro Expander. TTL.LIB, EPLDMAC.LIB, and INTEL.LIB are reserved and may not be used.

USERLIST is the name of the command file and must be preceded by the @ symbol. The command file is simply an ASCII text file that can be modified to contain any number of macros desired. MLIB processes the entire list of macros on each invocation. To add a new macro to an existing library, add the name of the macro to USERLIST, and create the new library by entering the command line shown above. Command file names follow DOS conventions. MLIB supplies a .DEV extension if no extension is specified. MLIB searches first in the current directory, then along the DEV environment variable, and finally along the PATH environment variable for the files.

In order to connect inut and output primitives, the files INPUT.DEV and OUTPUT.DEV must be included in at least one of the libraries. These files are contained in the TTL macro library.

Figure 4 shows a sample MLIB command file that includes options, the library name, and the names of seven macro files to be included in the library in addition to the INPUT and OUTPUT macros. The format of the command file is free form. Note that comments can be included in the command file and must be contained within percent (%) signs.

Note that the -l option cannot be included in an MLIB command file; it can only appear on the command line. The -l option lists the contents of existing libraries; it does not list library contents while building a library.

```
-o PROJA.LIB % macro library name %
-v
-s V1.50 % version number %
-c "Copyright (C) Date, Your Company, Your Name"
                % copyright information %
-d % display directory %

% include the following macros %

INPUT.DEV   OUTPUT.DEV   7408.DEV
7487.DEV    74138.DEV    74139.DEV
74151.DEV   74157.DEV    74251.DEV
                                    292040-4
```

Figure 4. Sample Command File for MLIB

The command line to process the file shown in Figure 4 is as follows:

MLIB @SAMPLE <Enter>

where SAMPLE is the name of the command file.

To list the contents of PROJA.LIB after creation, invoke MLIB as follows:

MLIB -l PROJA.LIB

This command line lists the macros in PROJA.LIB to the screen. The DOS file redirection capability can also be used to create a disk file listing the contents of macro libraries. For example:

MLIB -l PROJA.LIB > PROJA.DOC

SAMPLE SESSION: COMMAND DECODER USING MACROS

Decoding logic is one common function implemented by programmable logic devices. The target circuit for this example is a device that decodes microprocessor command signals in selected address ranges. The target application and decoder requirements are as follows:

- The target application is a 16-bit microcomputer system with 1-Megabyte of memory and about two dozen I/O ports.
- The memory is divided into shared memory (lower 512K bytes) and local memory (upper 512K bytes). Shared memory resides off the processor board and requires active low memory command signals. Local memory resides on-board and requires active high memory command signals.
- I/O ports are also split between on-board devices requiring active high signals and off-board devices requiring active low signals. I/O devices between the address range F000-FFFFH are on-board; devices below that range (0000-EFFFH) are off-board.

- All interrupt requests are resolved by an on-board interrupt controller. Therefore, only an active high on-board interrupt acknowledge signal is needed.
- On-board control signals are always high or low, never three-stated. Off-board control signals are three-stated when not being used to execute a bus cycle. An external bus arbiter accepts a request signal from the command decoder and, after gaining control of the bus, sends address enable and command enable signals back to the command decoder.

Figure 5 shows a block diagram of the application, including the target EPLD design. The three functional blocks to be included in the EPLD are highlighted (not shaded).

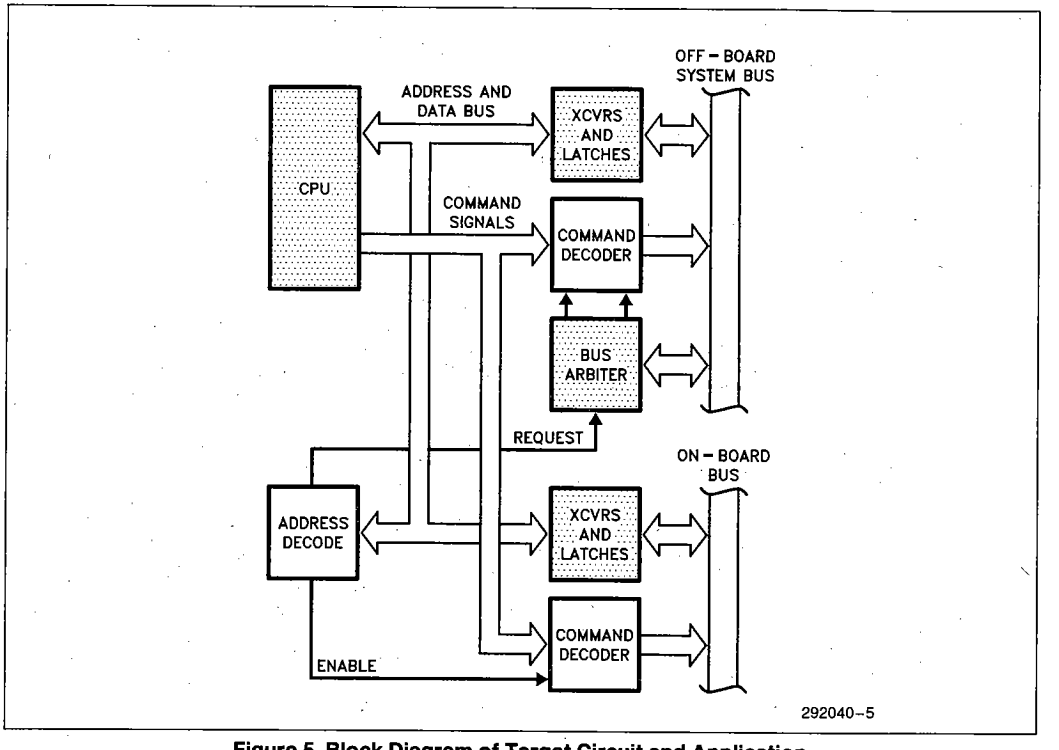

Figure 5. Block Diagram of Target Circuit and Application

Creating the Macro

Figure 6 shows a schematic diagram for the active low command decoder implemented with OR gates (low inputs enable the outputs; high inputs disable the outputs). Figure 7 shows the macro file that implements the circuit (CMDLO.DEV). This file was created with an ASCII text editor. Used as is, it provides the active low outputs for the design. With inputs RD, WR, and INTAIN inverted, it also provides the active high outputs for the design. This design uses CONF primitives to implement the three-state outputs in the macro. As an alternative, equations alone could have been used with the CONFs included in the ADF.

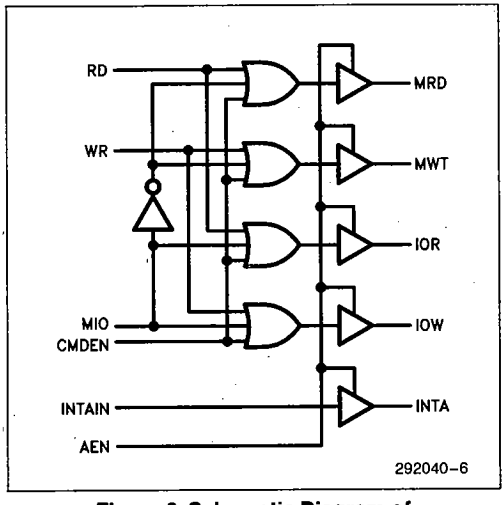

Figure 6. Schematic Diagram of Command Decoder

Building the Library

Use your text editor to create an MLIB command file that includes CMDLO.DEV, INPUT.DEV, and OUTPUT.DEV. The following example shows a sample command file named MACLIST.

```
-v  % show status %
-c  "1987, AP-312 Sample Macro Library"
-o  AP312.LIB
-d  % show the list %

% include the following macros %

CMDLO.DEV   INPUT.DEV   OUTPUT.DEV
```

Invoke the Macro Librarian with the following command line:

```
MLIB  @MACLIST
```

The Macro Librarian processes the three macro files and stores them in a user library named AP312.LIB. The library contains the copyright statement "1987, AP-312 Sample Macro Library". When processing is complete, MLIB returns control to DOS.

Creating the ADF

Figure 8 shows a schematic diagram for the target circuit. Figure 9 shows the ADF for the circuit (COMCODE.ADF), which invokes both instances of the CMDLO macro and contains equations used to enable the decoders under the proper conditions. The ADF signal named ONBEN (On-Board Enable) enables the active high decoder). The AEN (Address Enable) input to the on-board decoder is left unconnected. The default (always enabled) will be used.

```
CMDLO(MIO,RD,WR,INTAIN,CMDEN,AEN,MRD,MWT,IOR,IOW,INTA)
DEFAULT:(GND,VCC,VCC,VCC,GND,GND,,,,,)

NETWORK:

MRD  = CONF(MRDc,AEN)
MWT  = CONF(MWTc,AEN)
IOR  = CONF(IORc,AEN)
IOW  = CONF(IOWc,AEN)
INTA = CONF(INTAIN,AEN)

EQUATIONS:

MRDc = /MIO + RD  + CMDEN;
MWTc = /MIO + WR  + CMDEN;
IORc = MIO  + RD  + CMDEN;
IOWc = MIO  + WR  + CMDEN;

ENDEF
```

292040-7

Figure 7. Macro File for Command Decoder (CMDLO.DEV)

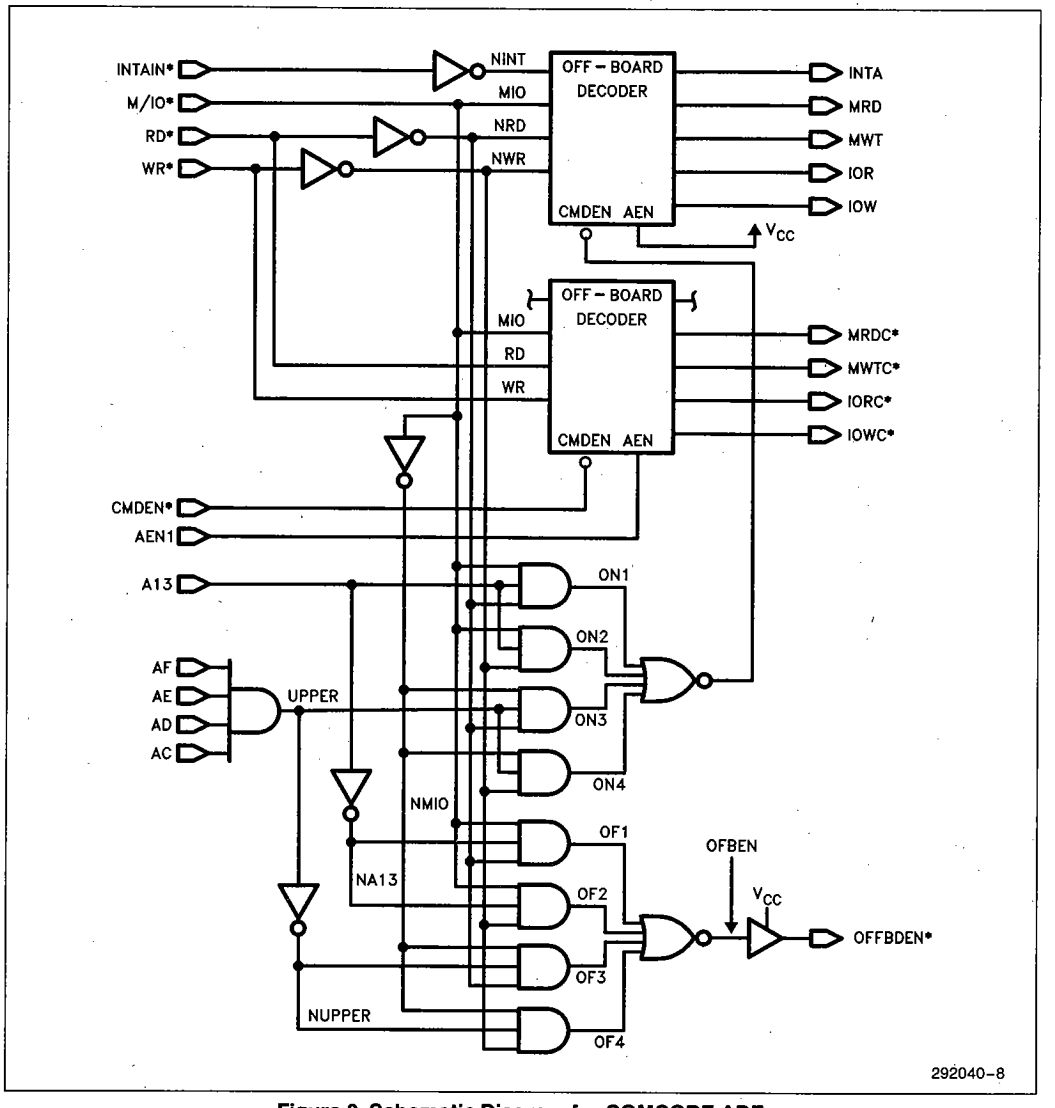

Figure 8. Schematic Diagram for COMCODE.ADF

```
DANIEL E. SMITH
INTEL CORPORATION
4/7/87
1
A
16209-001
COMMAND DECODER

OPTIONS:  TURBO=ON
PART:  5C090
INPUTS:  MIO, RD, WR, INTAIN, CMDEN, AEN1, A13, AF, AE, AD, AC
OUTPUTS:  MRD, MWT, IOR, IOW, INTA, MRDC, MWTC, IORC, IOWC, OFFBDEN

NETWORK:

INPUT(MIO,MIO)
INPUT(RD,RD)
INPUT(WR,WR)
INPUT(INTAIN,INTAIN)
INPUT(CMDEN,CMDEN)
INPUT(AEN1,AEN1)
INPUT(A13,A13)
INPUT(AF,AF)
INPUT(AE,AE)
INPUT(AD,AD)
INPUT(AC,AC)

OUTPUT(MRD,MRD)
OUTPUT(MWT,MWT)
OUTPUT(IOR,IOR)
OUTPUT(IOW,IOW)
OUTPUT(INTA,INTA)
OUTPUT(MRDC,MRDC)
OUTPUT(MWTC,MWTC)
OUTPUT(IORC,IORC)
OUTPUT(IOWC,IOWC)

CMDLO(MIO,RD,WR,,CMDEN,AEN1,MRDC,MWTC,IORC,IOWC,) % OFB %

CMDLO(MIO,NRD,NWR,NINT,ONBEN,VCC,MRD,MWT,IOR,IOW,INTA) % ONB %

OFFBDEN = CONF(OFBEN,VCC)
OFBEN   = NOR(OF1,OF2,OF3,OF4)
ONBEN   = NOR(ON1,ON2,ON3,ON4)
NRD     = NOT(RD)
NWR     = NOT(WR)
NINT    = NOT(INTAIN)
NMIO    = NOT(MIO)
NUPPER  = NOT(UPPER)
NA13    = NOT(A13)

EQUATIONS:

UPPER = (AF * AE * AD * AC);
ON1   = (MIO * A13 * NRD);
ON2   = (MIO * A13 * NWR);
ON3   = (NMIO * UPPER * NRD);
ON4   = (NMIO * UPPER * NWR);
OF1   = (MIO * NA13 * NRD);
OF2   = (MIO * NA13 * NWR);
OF3   = (NMIO * NUPPER * NRD);
OF4   = (NMIO * NUPPER * NWR);

END$
```

292040-9

Figure 9. ADF for COMCODE.ADF

OFFBEN (Off-Board Enable) requests permission to access the off-board bus from the external bus arbiter. The bus arbiter enables the off-board decoder via AEN1 (Address Enable 1) and CMDEN (Command Enable). CMDEN allows the appropriate signal to go high or low, and AEN1 causes the outputs to independently enter or exit a high impedance state (three-state).

Note the same name is used for both nodes of each INPUT and OUTPUT macro call. Use of the same name ensures proper connection when the Macro Expander eliminates redundant primitives (for example, a CONF feeding another CONF).

Compiling the Design

Proceed as follows to compile the ADF.

1. Include AP312.LIB in the IPLS environment variable. From the DOS command prompt, type:

   SET IPLS=C:\IPLSII\AP312.LIB; . . . <Enter>

   For user-defined macro libraries that are regularly accessed, the IPLS variable can be set in an AUTOEXEC.BAT file.

2. Invoke the iPLS II Menu by entering:

   IPLS <Enter>

3. Invoke the LOC from the Main Menu by pressing <F4>.

4. Answer the LOC prompts as follows:

Input Format?	<Enter>
File Name?	COMCODE <Enter>
Minimization?	Y
Inversion Control?	N
LEF Analysis?	Y
Error Message File	COMCODE.ERR<Enter>

   The LOC then asks:

   Do you wish to run under the above conditions [Y/N]?

   Enter: Y

   The LOC expands the macros and compiles the expanded file to produce a JEDEC programming file (COMCODE.JED), a utilization report file (COMCODE.RPT), a minimized logic equation file (COMCODE.LEF) and an error message file (COMCODE.ERR). For traceability, a file called COMCODE.SDF is created to show the expanded form of the ADF output by the Macro Expander.

5. The LOC terminates execution with the following message:

   LOC cycle successfully completed

You can examine the LEF file to see the minimized form of the design. The LEF shows the EPLD primitives used to implement the design. Macro calls are not shown in the LEF. If you wish, you can also use LPS (Logic Programmer Software) to program a part.